图文 093、案例实战：一次服务类加载器过多引发的OOM问题排查实践！

详情 评论

案例实战：

一次服务类加载器过多引发的OOM问题排查实践！

狸猫技术窝专栏上新，基于**真实订单系统**的消息中间件（mq）实战，重磅推荐：

未来3个月，我的好朋友原子弹大侠将带你一起，全程实战，360度死磕MQ

（点击下方蓝字进行试听）

从 0 开始带你成为消息中间件实战高手

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我都会逐一答疑

（ps：评论区还精选了一些小伙伴对**专栏每日思考题的作答**，有的答案真的非常好！大家可以通过看别人的思路，启发一下自己，从而加深理解**）**

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群。**

（群里有不少**一二线互联网大厂的助教**，大家可以一起讨论交流各种技术）

具体**加群方式**请参见文末。

（**注**：以前通过其他专栏加过群的同学就不要重复加了）

狸猫技术

进店逛

相关频道

从 0 开
战高手
已更新1

1、案例背景引入

公司里有一个非常正常的线上的服务，采用的是Web系统部署在Tomcat中的方式来进行启动的。

但是有一段时间，我们突然收到一些反馈，说是这个服务非常的不稳定，经常会出现访问这个服务的接口的时候出现服务的假死问题。

一旦出现这种接口调用时服务假死的情况，相当于我们的这个服务就完全不可用了，因此收到了不少上游服务的反馈。

但是上游服务反馈了另外一个非常关键的情况，就是经常一段时间内无法访问这个服务的接口，但是过了一会儿又可以访问了！

也就是说，似乎每次系统假死都只是一段时间而已！

因此我们着手进行了下面一系列的排查，最终解决了这个问题。

2、使用top命令检查机器资源使用

这里先给大家介绍一个技巧，因为当时的生产情况是服务假死，接口无法调用，并不是直接就抛出了OOM之类的异常。

因此其实你也很难直接去看他的线上日志，说根据日志立马就可以定位问题。

因此针对服务假死这个问题，我们首先可以先用linus的top命令去检查一下机器的资源使用量，通过这个命令可以看到机器上运行的各个进程对CPU和内存两种资源的使用量。

为什么要看这个呢?

因为如果服务出现无法调用接口假死的情况，首先要考虑的是两种问题。

第一种问题：这个服务可能使用了大量的内存，内存始终无法释放，因此导致了频繁GC问题。

也许每秒都执行一次Full GC，结果每次都回收不了多少，最终导致系统因为频繁GC，频繁Stop the World，接口调用出现频繁假死的问题。

第二种问题：可能是这台机器的CPU负载太高了，也许是某个进程耗尽了CPU资源，导致你这个服务的线程始终无法得到CPU资源去执行，也就无法响应接口调用的请求。这也是一种情况。

因此针对服务假死的问题，通过top命令先看一下，立马心里就有数了。

针对线上这台机器使用top命令检查之后，就发现了一个问题，这个服务的进程对CPU耗费很少，仅仅耗费了1%的CPU资源，但是他耗费了50%以上的内存资源。这个就引起了我们的注意。

因为这台机器是4核8G的标准线上虚拟机，针对这种机器通常我们会给部署的服务的JVM总内存在5G~6G，刨除掉Metaspace区域之类的，堆内存大概会给到4G~5G的样子，毕竟还得给创建大量的线程留下一部分的内存。

之前给大家介绍过，JVM使用的内存主要是三类，栈内存、堆内存和Metaspace区域，现在一般会给Metaspace区域512MB以上的空间，堆内存假设有4G，然后栈内存呢？每个线程一般给1MB的内存，那么如果你JVM进程中有几百上千个线程，也会有将近1G的内存消耗。

此时JVM进程其实耗费的总内存就接近6G了，另外你还得给操作系统内核以及其他的进程留出一部分的内存空间去使用。

因此最终让你的JVM可以使用的堆内存大概也就是机器上一半的内存而已。

大家到这里就知道了，为什么我们看到这个服务进程对内存耗费超过50%感到有点惊讶，因为这说明他几乎快要把分配给他的内存消耗殆尽了！

而且最关键的是，他长期保持对内存资源的消耗在50%以上，甚至达到更高，说明他GC的时候并没有把内存回收掉！

3、在内存使用这么高的情况下会发生什么?

此时我们开始思考一个问题，既然这个服务的进程对内存使用率这么高，可能发生的问题也就三种。

第一种是内存使用率居高不下，导致频繁的进行full gc，gc带来的stop the world问题影响了服务。

第二种是内存使用率过多，导致JVM自己发生OOM。

第三种是内存使用率过高，也许有的时候会导致这个进程因为申请内存不足，直接被操作系统把这个进程给杀掉了！

所以此时我们就开始分析，到底是哪种问题呢？

首先我们先使用jstat分析了一下JVM运行的情况，确实内存使用率很高，也确实经常发生gc，但是实际上gc耗时每次也就几百毫秒，并没有耗费过多的时间

也就是说虽然gc频率高，但是其实是个常规现象。

而且我们发现这个服务经常频繁gc，但是有的时候频繁gc的时候，也没听见上游服务反馈说服务假死！因此第一种情况其实直接可以过滤掉了。

接着我们分析第二种情况，难道是JVM自己有时候发生OOM挂掉了？挂掉的时候必然导致服务无法访问，上游服务肯定会反馈说我们服务死掉的！

但是我们检查了一下应用服务自身的日志，并没有看到任何日志输出了OOM异常！

接着我们猜测，也许是第三种问题，就是JVM运行的时候要申请的内存过多，结果内存不足了，有时候os会直接杀掉这个进程！

可能在进程被杀掉的过程中，就出现了上游服务无法访问的情况，但是我们的进程都是有监控脚本的，一旦进程被杀掉，会有脚本自动把进程重新启动拉起来。

所以也许其他服务过一会就会发现，服务又可以访问了！

4、到底是谁占用了过多的内存？

既然如此，按照我们的思路继续分析下去，如果要解决这个问题，就必须要找出来，到底是什么对象占用我们的内存过多，进而申请过多的内存，最后导致进程被杀掉了？

很简单，直接从线上导出一份内存快照即可。

我们在线上系统运行一段时间过后，用top命令和jstat命令观察了一段时间，发现jvm已经耗费了超过50%的内存了，此时迅速导出了一份内存快照进行分析。

此时用MAT进行内存快照分析的时候，我们发现，居然是一大堆的ClassLoader也就是类加载器，有几千个，而且这些类加载器加载了的东西，都是大量的byte[]数组，所有这些一共占用了超过50%的内存。

看起来元凶就是他了！

那这些ClassLoader是哪儿来的？为什么会加载那么多的byte[]数组？

具体原因就不说了，但是大家应该知道一点，我们除了用类加载加载类以外，其实还可以用类加载器去加载一些其他的资源，比如说一些外部配置文件什么的。

当时写这个系统代码的工程师做了自定义类加载器，而且在代码里没有限制的创建了大量的自定义类加载器，去重复加载了大量的数据，结果经常一下子就把内存耗尽了，进程就被杀掉了！

因此解决这个问题非常的简单，直接就是修改代码，避免重复创建几千个自定义类加载器，避免重复加载大量的数据到内存里来，就可以了。

5、本文小结

其实所谓的案例实战，他们背后的原理都是一套东西，归根结底从原理层面都是类似的

但是用不同的案例，可以告诉大家各种不同的故障场景以及不同问题的分析思路，这个是案例对大家最有用的一个地方。

因此希望大家好好吸收各种不同的案例，去体会不同场景的问题和解决思路。

End

如何加群?

添加微信号：Lvgu0715_ （微信名：绿小九），狸猫技术窝管理员

发送 Jvm专栏的购买截图

由于是人工操作，发送截图后请耐心等待被拉群

最后再次提醒：通过其他专栏加过群的同学，就不要重复加了

狸猫技术窝其他精品专栏推荐：

21天互联网java进阶面试训练营（分布式篇）